## At undervise i informatik

Kortlink [kortlink.dk/2ktm9](https://kortlink.dk/2ktm9)

Af Claus Witfelt

For mange unge er det første møde med at skulle udvikle noget/programmere i selv simple udviklingsmiljøer som Scratch, Code.org m.fl. informatik-faget i gymnasiet. Denne tekst er et forsøg på at rammesætte den didaktik, som jeg selv anvender. Teksten hviler kraftig på Michael Caspersen, Seymour Papert, Mitch Resnick, Emanuella Marchetti og Andrea Valente og andres ideer, se litteraturlisten.

1. En kreativ og udforskende tilgang, der tager udgangspunkt i elevernes egen livsverden.
2. Use-modify-create og Thinkering/”fusken”
3. Skridtvis forfinelse, faded guidance og det iterative designloop
4. Worked examples / Video-baseret læring
5. Engagement og embodied computing
6. Survival strategies

## Ad. 1. En kreativ og udforskende tilgang, der tager udgangspunkt i elevernes egen livsverden.

IT er et hverdagsfænomen, noget som gennemsyrer vores liv, arbejde, fritid og kommunikation. Som Marchetti/Valente påpeger, så tager rigtig meget it-undervisning et teknisk udgangspunkt, men skal man ramme bredt, bør man *ikke* tage udgangspunkt i fx hvordan man sorterer en liste af tilfældigt genererede tal (visse steder kaldet *Computer Science-traditionen* i modsætning til *Computational thinking traditionen*), men i stedet tage udgangspunkt i hverdagsfænomener fx noget, som unge forbinder med computere, Internettet og IT, mode, musik, shopping, kommunikation m.m.

Allerede i 1980’erne havde Seymour Papert set, det kæmpe potentiale der er i at anvende computeren som *skabende og kreativt* medie og at disse aktiviteter kunne danne en bro til en eksperimenterende tilgang til andre fag. Han opfandt programmeringssproget *Logo*, på dansk *MyreSnak* og Mitch Resnick tog skridtet videre og skabte en ekstremt lettilgængeligt, netbaseret og gratis version: *Scratch*, hvor børn og unge kan skabe programmer, musik, historier og meget andet vha. et motiverende og simpelt klodsbaseret programmeringsmiljø (http://scratch.mit.edu).

Et billede, der indeholder krybdyr, skildpadde, sten

Automatisk genereret beskrivelse Et billede, der indeholder tekst

Automatisk genereret beskrivelse

## Ad. 2. Use-modify-create og Thinkering/”fusken”

Selv for meget trænede programmører kan det være en svært at gå direkte fra et problem fx ”skab en webside, hvor en kunde kan logge ind og bestille en vare” og til direkte at skrive korrekt kode. En god strategi kan i stedet være use-modify-create, hvor eleverne først får lejlighed til at afprøve en løsning, som læreren (eller andre har udviklet), for derefter at kunne ændre lidt på den, udvide antallet af brugere m.m. for til sidst at slippe dem løs og skabe skabe egne løsninger.

I forlængelse af dette princip ligger det, som Papert kalder *Tinkering*, på dansk kan man måske oversætte det med *fusken*, *roden-med*. At arbejde med at udvikle it-løsninger er meget sjældent ”den lige vej fra en ide til man skriver kode, der virker første gang”, men ofte en proces, hvor man skal prøve sig vej frem, fejle, prøve igen, prøve en anden måde, teste også så videre. Professionelle it-folk taler netop om work-arounds om denne proces. Man kan forsøge at give eleverne nogle overlevelses-strategier – hvad gør de, hvis de går i stå, ikke kan komme videre. Dette gælder også for lærere, der underviser i IT – hav et sæt af overlevelsesstrategier, når man skal finde fejl i elevernes kode m.m. Her kommer abstraktion/mønstergenkendelse ind.

## Ad. 3. Skridtvis forfinelse og det iterative designloop.

Disse principper ligger i forlængelse af punkt 3. Iteration betyder gentagelse og processen med at udvikle programmer er ofte at skrive lidt kode, løse en lille del af opgaven, teste, tjekke at det virker, udvide opgaven lidt, skrive lidt mere kode og så fremdeles. Den iterative model, hvor fx virksomheder udvikler for deres kunder, hvor man hele tiden kører rundt mellem design, programmering og test, arver fra dette princip.

*Faded guidance* arver fra Vygotskis zonen for nærmeste udvikling, men er dynamisk, i starten meget støtte, efterhånden slippe eleverne mere fri, indtil de kan arbejde helt selv.

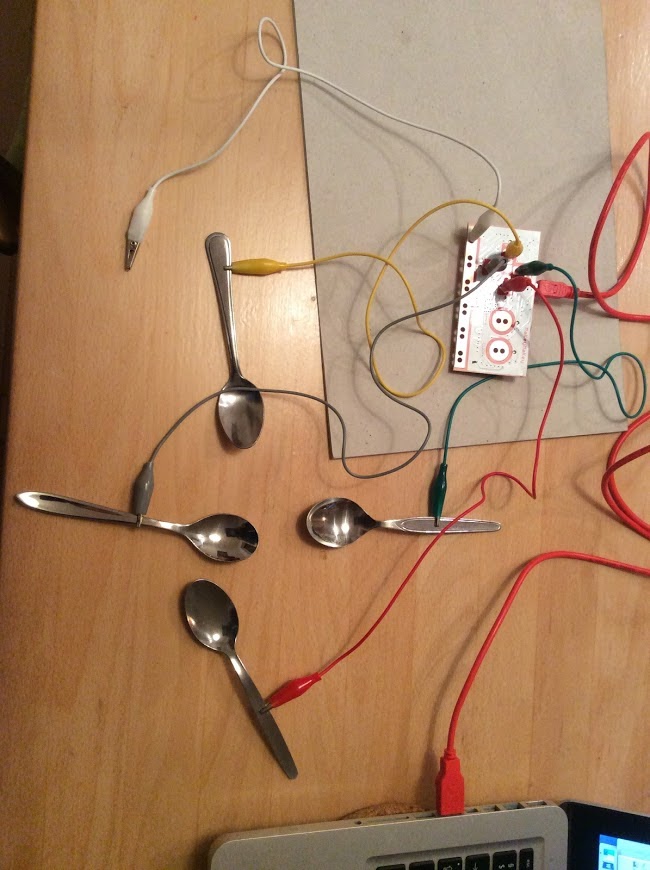
## Ad. 4. Worked examples

*Worked examples* er måske nok det allermest centrale princip. Mange er nok vante med, at hvis de skal løse et problem, fx lappe en cykel eller lign., så går man direkte uden om brugervejledninger og til på YouTube eller lign. video-kanal.

Et ’worked example’ er både et eksemplarisk problem og et detaljeret instruktion om, hvordan det løses. At se en lærer stå og udvikle programmer på en tavle er anstrengende og ikke særligt motiverende. I stedet er det godt at have et bibliotek af videoer, som eleverne kan arbejde sig igennem alene eller i par. Jeg benytter ofte at lave programmerne og speake ind over i ”små bidder” af 3- minutter maks. Eleverne kan så høre/se lidt, gør det samme, teste, rette, høre lidt mere og så videre. Man kan faktisk også undervise i teori på denne måde, som Daniel Shiffmann gør meget humoristisk (Shiffmann 2023). Det frigør lærerens tid så man kan hjælpe dem, der er kørt fast i. Højst 5% lærer-ved-tavle og 95% elever arbejder med worked examples og ”tinkering”.

## Ad.5. Engagement, spil og embodied computing

At arbejde med IT, kreativitet og sociale medier er sjovt og det skal gerne blive endnu sjovere, når man begynder at fremstille små-programmer, apps og løsninger, noget, Seymour Papert også betoner i hans bog ”Mindstorms” – på dansk ”Den totale skildpaddetur” (Gads forlag 1983). Netop kreativitet, det skabende og fusker-tilgangen kan både være engagerende og frustrerende. En udvidelse, som jeg selv benytter til at gøre IT sjovere er at lave små-spil og arbejde med *embodied computing*, fx at tilføje Arduino-kloner, robotter og anden elektronik til, for at gøre introduktion til computing sjovere, stimulere lysten til at ”fuske” m.m.



Listen kunne fortsættes her med it-genrekendskab og meget mere.

## Litteraturlisten

* Caspersen, Michael: “Fra teknologiforståelse til informatik” MONA
* Marchetti/Valente: “Simplifying Programming for Non-technical Students: A Hermeneutic Approach”, <https://pubmed.ncbi.nlm.nih.gov/35068698/>
* Resnick, Mitch: ”Let’s teach kids to code”, <https://www.youtube.com/watch?v=Ok6LbV6bqaE&t=279s&ab_channel=TED>
* Papert, Seymour: “Den totale skildpaddetur : børn, datamaskiner og kreative tanker”, Gad, København 1983
* Witfelt, Claus: ”Informatik for alle” <https://informatikforalle.ibog.forlagetcolumbus.dk/>
* (Shiffmann 2023). Daniel Shiffmann: ”The coding Train” <https://www.youtube.com/channel/UCvjgXvBlbQiydffZU7m1_aw>